Intro to Algorithm Analysis

• Algorithm analysis
  – worst case
  – big-O
  – $O(n), O(1), O(n^2)$

• Big-O of **merge** method
  – Merge algorithm example
  – finding big-O of Java library methods

• Big-O of **Sequence** class
Announcements

• Lab this week: bring your laptop with Eclipse installed (or use Eclipse on lab computer)

• Midterm 1 is this Thu 2/18
  – Location: THH 101
  – Time: 8am – 9:20am (normal lecture time)
  – Closed book, closed note, no electronic devices
  – Bring USC ID card

• Reminder: academic integrity
Algorithm analysis idea

• Compare one algorithm / data structure to another before implementation.
• For a given problem size $n$, how long does it take?
  – worst-case performance.
  – average-case performance.
Algorithm analysis idea (cont.)

- Ex: “search an array”:
  - does the value, target, appear in a given array of size $n$, and if so, at what position?
- Want to know how long it takes as a function of $n$.

$target = 12$

<table>
<thead>
<tr>
<th>values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>14</td>
<td>9</td>
</tr>
</tbody>
</table>
Big-O notation

*asymptotic* worst-case performance:

- behavior as $n$ grows,
- on worst possible input of size $n$.
- Use *big-O* notation.
- units are program steps.
  - e.g., $2 * 2$ takes the same amount of time as $20000 * 20000$
O(n)

• Example 1: It takes $n$ steps to print all the elements in an array with $n$ elements.
  – We say this algorithm is “order $n$”,
  – or $O(n)$, or
  – it takes “linear time”.

• $O(n)$ means number of steps is some linear function of $n$:
  $$c_1 \cdot n + c_2$$
Counting steps
Example 2: compute the average of $n$ numbers:

```java
int sum = 0;
int n = in.nextInt();

for (i = 1; i <= n; i++) {
    int value = in.nextInt();
    sum += value;
}

System.out.println("The average is: " + sum / ((double) n);
```
Why ignore constants?

- constants fade away as $n$ grows large.
- compare algorithm to another that may differ in order of magnitude, e.g., $O(n^2)$ or $O(2^n)$
- distinct from “tuning” a specific implementation
Algorithm that takes the same amount of time no matter how big \( n \) is.

- called *constant time*
- or *order 1*
- or \( O(1) \)
O(1) examples

• Examples:
  – assignment statements
  – arithmetic expressions
  – comparisons
  – println
  – simple-statement sequences
  – loops with constant bounds

• Ex:
  – Input: an array of size n.
  – Problem: find the $k$th element in the array.
Sequential search

- Ex 3: Big-O to search in an unordered array of size $n$.
  - does the value, $target$, appear in a given array of size $n$, and if so, at what position?
  - time depends on values in array.
  - we’re interested in the worst case.

<table>
<thead>
<tr>
<th>values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>14</td>
<td>9</td>
</tr>
</tbody>
</table>

$target = 12$
Sequential search on sorted array

- Ex 3 (variation): Big-O to search in an ordered array of size $n$ using linear search (i.e., F10 MT1 problem)
- Worst case?
- best case, average case?

$$target = 8$$

<table>
<thead>
<tr>
<th>values</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>14</td>
</tr>
</tbody>
</table>
• Ex 4: print out a multiplication table for the integers 1 to $n$

• *Quadratic time* ($O(n^2)$) is any quadratic function of $n$: $an^2 + bn + c$
Ex: merge two ordered lists

• problem: create one large ordered ArrayLists out of two ordered ArrayLists (no duplicates).

• Example:
  – list1: 3 7 9 12 15
  – list2: 2 5 6 8 9 20
  – merged list: 2 3 5 6 7 8 9 12 15 20
slow `merge` method

- Idea: for `merge(list1, list2)`:  
  - copy arraylist in `list1` to `result` arraylist (copy constructor)  
  - for each element of arraylist in `list2`:  
    - find its location in `result`  
    - insert the element at that location in `result`  
      (use `ArrayList` method `add(index, elmt)`)  
  - return `result`  

- big-O? (size of `list1` is `m`, size of `list2` is `n`)
Better-performing *merge* method

- take advantage of the fact that both arrays are already sorted
- traverse both arrays in one loop:
  - take the smaller element of the two arrays and add it to the result array, and update index of the one moved.
- every loop iteration get closer to the end of one of the arrays
- always adding new values at the *end* of result
- *merge algorithm*
Merge example

List1:  9   11   16   20
List2:  2   5   16   17   18
Comparing different time bounds

• Sometimes there exist fast and slow algorithms to solve the same problem.
• Here’s an idea of what some of these time bounds look like when plotted.
big-O practice

• **Sequence** class:
  – to represent a sequence (list) of numbers

• **Internal implementation:**
  – values are stored in an array (beginning and end of array corresponds to beginning and end of the sequence)

• **Operations next page . . .**
# Sequence class operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>big-O (for array rep)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence <code>s = new Sequence();</code></td>
<td></td>
</tr>
<tr>
<td><code>s.getValAt(loc) → val</code></td>
<td></td>
</tr>
<tr>
<td><code>s.contains(val) → t/f</code></td>
<td></td>
</tr>
<tr>
<td><code>s.removeValAt(loc) → success</code></td>
<td></td>
</tr>
<tr>
<td><code>s.insertAtEnd(val)</code></td>
<td></td>
</tr>
<tr>
<td><code>s.insertInFront(val)</code></td>
<td></td>
</tr>
<tr>
<td><code>s.numVals() → length</code></td>
<td></td>
</tr>
</tbody>
</table>